Databases for Linguistic Purposes:
a case study of being always too early and too late

Peter Wittenburg, Daan Broeder, Richard Piepenbrock, Kees vd Veer

At the MPI for Psycholinguistics “databases” were in the center of interest as long as it exists, since experimental and observational data is the basis of all its empirically driven research and “databases” are necessary to organize and administer research. This usage of the word “databases” already may indicate its different usage and indicate the large variety of data sets that is associated with it. In this paper we do not want to give a definition, but first describe and analyze four concrete examples of data sets that are used in the MPI that required modeling and implementation decisions. Then we will summarize our experience and come to some general statements.

1.0 Database Examples

1.1 The TGORG Database

The first example will refer to a database that is still in use and the backbone for the support of the research since 1985. It is not a typical linguistic application, but discussed here as an excellent example of an rDBMS application.

One of the first data sets that was discussed broadly and that had a relevance for more than one or a small group of researchers was the TGORG database. It is a database used by members of the Technical Group to contain all its organizational data. It was extended during all the years. At the beginning the purpose was to administer the increasing amount of equipment that is in use by different people and groups in all aspects. In this database we had basically to deal with equipment units and persons (see appendix A). Each of these was to be described by a number of attributes. For “person” (PERS) one has the classical information types such as name (USER_NAME), group (GROUP_CODE), start of affiliation (FROM_DATE), end of affiliation (TO_DATE), status (USER_STATUS), telephone number (PHONE) etc. For “equipment” (EQUIPMENT) one gets a more complex structure, since a unit (INV_NR) has attributes such as equipment type (EQUIP_TYPE) and part of an order (ORDER_NR), but each type is associated with a model (EQUIP_MODEL) that has its own attributes (CAPACITY). The person and equipment information is linked for example by the user name (USER_NAME) or room number (ROOM_NR).

An analysis of the data model revealed that it fitted perfectly well with the Entity-Relationship model that came up at that time with its great advantage that all data (at least for the TGORG type) can be retrieved in flexible ways by applying simple algebra. We were convinced that we should start using relational databases and gather the necessary experience. First, a careful design of the logical structure of our data was made to prevent redundant data storage and to constrain were possible the value range of the fields. Also the possible use cases were studied to achieve performance for certain query types. Second, a careful comparison between relational database management systems such as ORACLE, INGRES and SIR[1] etc, i.e. software that allows to create, maintain and query relational databases, was made. For reasons that we don’t understand anymore we have chosen the right system for this task. ORACLE provided us a stable framework for about 20 years and we extended it at various moments to adapt the database to new needs. New information types were added: tables introducing new entities such as network aspects (HUB, HUB_STATUS, IP, IP_ET, PATCH_ROOM, ...) that can be easily linked with the existing ones and various attributes that enrich the description of the entities.

When designing and implementing the TGORG database we used many of the features the emerging software offered. The Data Definition Language of ORACLE allowed us to define various constraints, require uniqueness, execute “delete cascade” operations and others more. Database-Pre-Triggers[2] and Post-Triggers were used to ensure the consistency of the database. A typical Database-Pre-Trigger is to write records to a history table before executing an update. Triggers are also used for example when executing an update of an attribute that can occur as a key in several tables. Here Pre-

[1] Basically a hierarchical database with a relational shell which at that time was a popular combination.

[2] Database-Triggers are associated with attributes of a table and have to be distinguished from Application Triggers that are associated with forms. Both are supported by professional rDBMS.

Triggers are typically used to find the attributes, while post-triggers are used to clean up the temporal storage used. Two aspects were encountered at that early time: (1) The hope that colleagues will create and execute SQL statements as line commands turned out to be an illusion. Many of them were not eager or capable to understand the underlying algebra. At a certain moment it was possible to create forms for ORACLE which is now being supported at the user interface. (2) We needed a database manager – a specialist – to create and modify the database, the necessary user interfaces, the indexes, etc. (3) It was our experience that running a professional and complex database system requires much efforts for typical maintenance tasks as version upgrading or changing to new platforms.

Summary
The system was extended such that today several members of the TG using different views are operating on TGORG, that it is the central repository for all our administrative and organizational data, that it enables just one person to equip and administer about 30 expeditions with lots of small parts every year including the easy creation of lists for the customs of various countries, that central controllers can get the required information about all bits of purchased equipment and that we can do our equipment planning.

Given the relevance for giving a good and smooth support for the researchers over many years this database can be seen as a very positive example of applying relational databases. Also the early decision to hire one specialist to deal with all database aspects and who knows all the relevant details of ORACLE was a very good one[3]. Together with the decision for ORACLE it served for continuity and stability in many applications. ORACLE was also used to completely administer a scientific journal and for many smaller and larger linguistic applications (see below).

Nevertheless, the central control people argued in 1987 that we spent money for solutions where others just use files. So we were too early with relational databases. Only later they found out that we were the only institute that had a stable information system and then mentioned it as a very positive example.

1.2 The CELEX Database

In 1985 the Dutch Organization for Scientific Research wanted to foster the foundation of national expertise centers for the creation and maintenance of infrastructures that were considered to be of vital importance to the research community. In the area of languages two institutes were established: the Center for Lexical Information (CELEX) and the Speech Processing Expertise Center (SPEX). The goal of CELEX was the creation of consistent and comprehensive computer lexica for Dutch, English and German. These lexica were to contain all essential formal linguistic phenomena except definitions and add psycholinguistically relevant data such as frequency of usage.

The intended focus on computer-based lexica was seen as a necessary step to steer away from traditional lexicographic approaches and effect a breakthrough in fundamental and applied linguistic research. Together with a few other research institutes, the MPI submitted a proposal that contained a project formation plan and an argumentation for implementing the lexica by applying the relational paradigm. This plan was accepted, although we received a lot of criticism from linguists in particular who did not believe that the required linguistic complexity could be captured with the help of such simple mechanisms as relational databases.

In CELEX about eight linguists and computer scientists were to collaborate on creating the lexica. It was scheduled that after having bootstrapped the lexica from a number of source materials such as electronic text corpora and word lists about five years were necessary to complete the task. It appeared that there were two aspects that convinced the decision body to accept the proposal that was mainly formulated by MPI members – at least in its technical aspects. First, a very clear and strict project structure was proposed and second, we suggested choosing a relational database management system, again ORACLE to be precise, as the “holy core” of the lexicon construction efforts.

[3] Of course, this database specialist did not only take care of database applications, good backups and the proper upgrade to new versions, but also carried out many other programming tasks.

At the beginning a core logical structure for the lexical database was defined in collaboration with database experts from the Technical University in Eindhoven (see appendix B[4]). Although the original design was adapted several times, it was one of the strengths of the CELEX project that everyone was forced to make their wishes explicit – there was no room for ambiguities with respect to the lexical structure. For many linguists it was a new, but also challenging experience to agree on one structure. Some compromises had to be made, such as the realization that full normalization of the data could not be achieved and some redundancy had to be accepted. This had to do with the fact that linguistic data are never completely regular, and also with the practical consideration of the trade-off between potentially slow on-line dynamic derivation of features and faster retrieval of stored records. It was a major asset to put structure standardization at the center, since it was accepted by everyone that it could not be subject to too frequent changes. So after the design phase all emphasis could be placed on the development of the linguistic and technical components.

The result of the analysis of the logical structure was a set of for instance about 20 tables for the Dutch lexicon comprising all fields, their relations, their types, and their consistency criteria (implemented as database triggers and constraints). The Dutch lexicon for example contains about 400,000 full-forms, i.e. lexical entries. Further, the necessary operations for the creation, management and exploitation processes were designed. At that moment ORACLE did not have any string matching feature for text fields that was sufficient for us, so we came to an agreement with ORACLE to enable us to compile our own flexible string matching function into the ORACLE kernel in order to also achieve the necessary speed up. Since many people contributed with information such as morphological inflections and frequency counts, special lexicon output had to be generated and fairly complex programs were necessary to integrate all delivered information into a new lexicon version and to carry out the necessary consistency checks. Further, many programs had to be developed utilizing SQL embedded in procedural algorithms to allow the CELEX workers to generate word derivations, to check correctness, and also to simplify and validate data entry through the use of ORACLE Forms.

CELEX was sold in various formats to companies such as Philips, Microsoft, Apple, Hewlett Packard, Siemens, IBM and Rank Xerox. Since CELEX had to be offered interactively via the emerging Internet (not the web) to the Dutch research community, software based on the Unix Curses library was developed that supported quasi graphics on alpha-numeric terminals. At the end of the project it was decided not to further develop CELEX anymore and to convert it into formats that can be easily offered to interested people. CELEX is still available on CDROM from the Linguistic Data Consortium (LDC), with a set of simplified tab-delimited plain text tables, and can also be approached via a web interface with limited functionality (www.mpi.nl/world/celex). It is still used by various researchers.

During the process of gradually releasing our data to the public, the main problems we encountered were managing user accounting, the then limited storage capacity for derived data, i.e. user-defined lexica saved for further processing, slow retrieval speeds over the internet, and setting terminal emulations for various operating systems. We also ran up against too high processing loads for the complex computation of psycholinguistic concepts such as neighbors or uniqueness points, which both required setting off each selected lexical entry against the entire vocabulary.

Summary

After the CELEX lexica were ready a very handy computational linguist – someone who is very skilful in writing Perl scripts – argued: Why all this data encapsulation within a relational DBMS? Why not from the beginning stick with manipulating a couple of plain text files with useful scripts? Who would like to dare to argue with him?

Well, of course a single computationally competent linguist could work this way. However, if a team of people is to collaborate on a complex project like CELEX, it is hard maintaining file integrity and version management without the aid of a sophisticated data administration system like an rDBMS. Also, a database system comes with handy tools like automatic sorting, algorithmic tools like

[4] Appendix B1 shows the core of the production lexicon and appendix B2 an extended table layout. In CELEX we distinguished between a development and a production lexicon, both contained in the ORACLE DBMS. The development lexicon had many more tables to include more information. Reduction processes were used to map the information to the production version.

summation, data export and import mechanisms, data entry forms, report generation and so on, that otherwise would require additional programming efforts.

Finally, CELEX was one of the most successful lexicon projects, at least in Europe:

- It served many researchers for years as a research instrument.
- It served the industry to get good base material.
- At least the Dutch lexicon was very comprehensive, fairly consistent and nearly free of errors (lexica for other languages were less ambitious in scope and feature sets).
- AND we demonstrated to at least some linguists that lexicon information can indeed be represented with the help of relational tables.

So when CELEX started out, we were too early in a way, since linguists were not yet familiar with presenting lexical data in an Entity-Relationship (ER) model or even with computers at all. At a later stage we were too late, since web-based applications had taken over from character-based terminal interfaces very quickly, and many linguists had familiarized themselves with computational procedures to exploit data from plain text files or even straight from text corpora readily available from the web.

1.3 The Speech Error Database

In 2002 we got the request to bring various speech error databases together and offer the merged information via a simple web-interface. The statistical analysis of speech error and correction patterns is very informative to analyze speech production and self-monitoring processes. In so far speech error analysis can contribute to the knowledge about our brain's speech faculty.

There was no master plan behind gathering such speech errors, for a number of psycholinguists it seems that writing down speech errors (often with hand-writing) is a kind of hobby. The consequence is that every researcher used his/her own sets of attributes – they overlap only partly. Together with our collaborators in Utrecht who studied the nature of the descriptions and the semantics of the attributes, an XML Schema was developed (see appendix C) that represents all information from the different databases. Based on this Schema converters were developed to produce one validated XML file (see appendix D for an example of an entry). This file contains currently 8600 speech errors.

The question was now what should be done to come to a useful web-interface and to support acceptable query times. Two options were given: either importing the XML file into a native XML database or use the "good old" relational database paradigm. Earlier experiences from tests with a few native open source XML databases such as eXist did not show convincing results. Either they showed many bugs or they did not show any significant speed improvements compared to processing text files[5]. Since we have an ORACLE license also the new ORACLE version 9i was an option at that moment which claimed to offer also native XML database support. However, even with the help of ORACLE specialists it was not possible to import the speech error XML file. Due to the given time constraints we stopped these tests, designed a logical structure for a relational database and imported the XML file to ORACLE. So we were faced with the problem to transfer the XML Schema into a logical structure for a relational database and to import the XML file.

Since this database is a good case for studying some of the problems that can occur when converting an XML structure into a relational database design, we will discuss the structure in more detail. The Speech Error Database ("sedb") can have several corpora ("corpus") and each corpus can have several sub-corpora ("subcorpus"). Each sub-corpus can have many speech errors ("se") which is the basic unit of description and of analysis. Each speech error can have a number of interpretations as essential results of the linguistic analysis. For each error group included in an interpretation there can be several linguistic analysis types. Also interesting in this respect is the fact that there are many attributes that can occur several times, i.e. there are many 1:N relations (see appendix C). In a relational database this would lead to separate tables. In addition, such a merged database includes many fields with "Null" values, i.e. missing information and often data such as speaker name, date or the correction string are not known even if they were used by a certain researcher. So the Speech-Error-Database is only sparsely filled.

[5] We tested such XML databases with collections of 10.000 XML documents. The systems were much too slow at that time.

This sparse filling of the speech error base and the fact that only a small number of fields are relevant for searches led us to not apply a strict method such as the object-relational mapping, but follow a hybrid and hand-crafted approach: Only those attributes that were expected to be subjects of searches were included explicitly as attributes in tables. The remaining elements of the XML file were put as XML-chunks into a special attribute. During the presentation of hits this XML information is visualized as HTML. A joint analysis with the responsible researcher led us to a comparatively simple table design where the error and its linguistic analysis is in the center. The key linking the relevant tables together are the error-id (ERROR_ID), the interpretation id (INTERPETATION_ID), errorgroup id (ERRORGROUP_ID). To link the error-type which is the result of the linguistic analysis with procedure information the analysis id (ANALYSIS_ID) is also used.

All other information is simply preserved as XML/HTML information, such that if a hit is found simply this information can be presented. Since joining many small tables at query time can be very inefficient the presentation of a chunk of text costs just one access, i.e. we expected a performance gain. A Perl script with embedded XML parsing and SQL components was used to import the XML data into the table structure.

Summary
XML seemed to be the natural choice to describe the structure of the merged database generated from different sources with different types of information included. To support fast searches a different container had to be used that promises to store the code more efficiently. Since only a few attributes are relevant for searching and since a systematic transformation of the XML Schema to a logical database structure would have yielded a large number of tables making the programming a larger effort a quick&dirty method was chosen to come to a database design. Basically this database was only meant for querying and visualization. We assume that the various researchers will continue to use their individualistic ways of describing speech errors, i.e. efforts have to be taken to merge other databases.

It was evident that we were too early to apply XML databases which would have been the choice at hand. Therefore, we stepped back to the good old relational technology. However, the nature of the database indicates some of the drawbacks of the ER model and a compromise was made to not get an explosion of useless tables.

1.4 The Metadata Database

While all databases discussed so far have a centralized storage concept, the IMDI database is a truly distributed one. Metadata descriptions adding to the IMDI database can reside on “central” sites, on project PCs or even on notebooks of individuals. The distributed IMDI database exists of many separate XML files linked together by corpus nodes (also XML files). When starting the IMDI enterprise we wanted to make the threshold for people to participate as low as possible. To form a controlled and uniform IMDI domain, of course it is necessary that all adhere to the IMDI schema (see appendix F). The best way to guarantee this is to offer an easy-to-use and professional editing tool. It should also be noted that IMDI MD is not only used for discovery but also for management purposes. With the help of IMDI a virtual structure can be created that organizes all resources and bundles them according to the wishes of the users. In this sense it can be compared with a distributed file system.

It is known that the OAI model for metadata discovery is fairly different in so far as it does not make any assumption about the individual holdings of the data providers and the metadata sets they are using. It defines a protocol according to which data providers have to offer metadata. In this domain metadata is only used for discovery purposes, a concept which has its strengths.

For IMDI we wanted to create a flexible, simple, human-readable and open domain of files which everyone interested can access and use. For browsing in this domain nothing special has to be done. To support browsing on the linked XML files we have developed an IMDI browser. For those who hesitate to download and install Java programs we also offer on the fly XSLT transformation to HTML allowing to browse in the IMDI domain with normal web-browsers. Needless to say that other services can be coupled with the IMDI domain such as access management and URID resolving[6].

[6] An access management system was already implemented based on the IMDI metadata. This allows to specify procedures and access rights for every node and its sub-sequent resources.

Of course, searching in such a distributed database is a time consuming task. Traversing the tree during query time and interpreting the IMDI XML text files would yield long reaction times. Therefore, as in the case of OAI service providers the XML files are read and converted into optimized containers to support fast searching – either Google-like simple search or complex search offering the elements. We have tested several solutions. The requirement that a metadata search solution would work with client applications such as the IMDI Browser without having to install or maintain special DBMS software limited our choices. The first solution we tried and that worked reasonably well for small number of metadata records was based on simple text-based index files and Perl scripts operating on them. However, they did not scale well. Currently, we use the Java RDBMS HSQLDB, it is embedded in client software such as the IMDI Browsers so users can use it for their private collections but is also used server side where it answers queries for metadata contained in the MPI corpus archive. The transformation from XML to rDB was made according to the object-relational mapping scheme resulting in a table structure as described by the Data-Definition-Language listing in appendix G. We found out that this solution scales much better with increasing numbers of records.

In the ECHO project where we were confronted with more than 150.000 records and completely different metadata schemes one requirement was a full text search option. We implemented a binary-tree based index solution where a given query word is represented by a sequence of nodes in this tree. At every node there are a number of vectors containing all relevant information for filtering. The processing time is largely independent of the number of records, i.e. the solution scaled well.

Whatever we are doing we have to keep the additional effort for linguists small. Since we have to support mixed searches on metadata and annotations we are still looking for the best solution.

Summary
In the IMDI domain the primary data structure are linked XML files, files that can be located somewhere on physical servers. In general this solution is excellent since it supports the openness of the IMDI domain. Everyone interested can create his/her own virtual linked structures and everyone can create his own services. However, for searching special measures have to be taken that are similar to the metadata harvesting OAI concept. Here, relational databases can play a role as secondary data structures supporting fast searches. Having chosen relational databases as primary container would have introduced unnecessary complexity: (1) rDBMS would have encapsulated the data, i.e. any form of access would have required special mechanisms. Using XML as the primary storage format ensures application and platform independent archiving. (2) Every user would have to have a relational database installed on his/her machine. (3) The combination to new virtual domains would have become more complex.

On the other hand it is a disadvantage at this moment that either the IMDI-browser has to be downloaded to operate on the XML files or an XSLT style sheet has to be associated with a web-server. So, whatever we do some downloading is involved. For the users we are too late at some point with easy solutions, for a seamless solution empowered by a fully operational web-service infrastructure we are too early.

2. Comparison

When we want to compare the two major options (the XML and the ER domain) we have to first distinguish between layers. For the domain of data management we miss a widely accepted reference model such as the ISO OSI model for computer networking. This OSI model allowed people to associate components of specific network protocol proposals or solutions with OSI layers and therefore to compare their functionality and usefulness. For our comparison we distinguish between data models as the conceptual layer (domain), documents as singular non-encapsulated formats, collections of such documents and special encapsulating containers which are called database management systems. We will not speak about content management systems, since they don't have a clear underlying data model and are containers that try to integrate collections and database management systems.

In the following table where we will compare the domains topic-wise we will indicate the different layers where applicable.

Topic	level	relational domain	XML domain
data model	domain	a set of related tables where entities are described by a set of attributes	basically a tree structure
implementation	document	tables in a plain text file with delimiters to indicate columns	can be native XML documents as plain text files
	collections	practically not existing	set of XML documents each of which is referenced by a URI
	DBMS	encapsulated tables	native XML DBMS
structure specification	document	no standardization for external explicit specifications	described by DTD or XML Schema standards
	collections	practically not existing	mediated by OS etc
	DBMS	described by DDL – stored as tables itself	no standards
cross-referencing	domain	not defined	Xlink standard
data typing	document	not applicable	defined by XML Schema
	DBMS	existing	defined by XML Schema
human readability	document	not standardized	yes
	DBMS	encapsulated representation – no	encapsulated – no
archivable format	document	not evident	yes
	DBMS	no	no
interoperability	document	not guaranteed	at encoding and syntactical level yes
	DBMS	via export – in fact no	via XML export - yes
distributed database	collection	possible - shares shortcomings of implementation	yes
	DBMS	special expensive systems yes	not yet I assume
correctness	document	no	validation by schema
	DBMS	yes – due to constraints and trigger usage	not clear
multi-user	document	not	not
	DBMS	yes – record locking etc	yes
integrity	document	no	no
	DBMS	yes – transaction, locking	yes
security	documents	by OS	by OS
	DBMS	additional	additional
user interface	document	normal editor	normal editor
	collections	special efforts	special efforts
	DBMS	excellent support	not clear
searching options	document	special efforts	Xpath, Xquery
	collections	special efforts	special efforts
	DBMS	SQL	Xpath, Xquery
searching speed	document	poor	poor
	collections	poor	poor
	DBMS	excellent due to optimizations and internal indexing	not as good

special searching aspects	DBMS	handling of texts not as good	
across data base search	collections	special efforts	special efforts
	DBMS	SQL	Xpath, Xquery
views	DBMS	native support	not clear
order	DBMS	not applicable	available
insert, update, delete	DBMS	formalized	yes at document level

In the literature we can find the distinction between “data-centric” and “document-centric” documents. “Data-centric” documents include highly structured data with a fairly regular structure and a high degree of granularity (fine-grained) where the order of the data is not relevant. “Document-centric” documents, however, have a much less regular structure, are less granular, contain mixed types and are designed for human consumption where the order of elements in most cases is relevant. “semi-structured” data can show regular structures, but they contain so much varation that a large number of columns (in table structures) would contain NULL values.

It is argued that for “data-centric” data the relational domain is most appropriate, although XML can also be used as interchange format for this type of data. For “semi-structured” and “document-centric” data it is argued that XML is the natural way to represent them. The Speech-Error-Database is a good example for semi-structured data.

3. Conclusions

After having described some concrete projects where we came to different solutions and after having compared the two domains, we now want to draw some general conclusions based on our experience in data driven projects.

1. Technology offers a wide variety of solutions to structure and represent data, each of them is supported by increasingly improved software. This software allows us to easily create the required presentation formats so that we can abstract from the underlying representation formats. It seems that we can get appropriate solutions for almost any problem we have – at least given the boundaries of our current programming paradigm.
2. A given data set seems to fit more intuitively to one of the major data models. Data-centric data seems to be optimally represented by the ER-model and document-centric by the tree-structure underlying the XML data model. However, there is no clear distinction between these two data types and the use of XML as an exchange format between different rDBMS formats makes clear that XML can and is also be used to represent data-centric data. Vice versa rDBMS give support for document-centric data.
3. There are differences in speed and ease with which individuals can describe their data set by using one of the two models. Some people start to use more abstract models such as UML to describe their data set. For example in ISO workgroups such as TC37/SC4 is a tendency to model data with the help of a modeling language such as UML and to see relational databases or XML-structured documents just as different instantiations of the underlying abstract data model.
4. When people speak about the advantages of the relational domain they often refer to the special features of relational database management systems such as transaction mechanism, data integrity, data security, multi user access, cross database actions, speed of searching and in particular the easiness with which it is possible to create simple user interfaces. However, creators of XML DBMS work hard to offer similar functionality.
5. XML is widely accepted as an interchange format. The reasons for this are that it does not require any special tools to interpret the data except those that come standard with the operating systems or de-facto standard web tools, that its structure is explicit, machine readable and also human readable, that it does not use any non-standard data encoding and encapsulation methods and that the schema is explicit, machine readable and human readable[7] as well.
6. Methods to convert between the two domains have been worked out such as the table-based or object-relational mappings that allow to transform an XML document into a relational database. However, specific information such as encoding and order information can be lost during such a transformation if no special measures are taken.

There is one remaining point to be discussed: Relational database management systems such as FileMakerPro, Access etc come with very simple to learn user interfaces and in between many linguists understand how to apply the ER model for their purposes. This results in many applications world-wide where linguists create databases with very interesting and relevant data. They are not aware that they encapsulate their data which is comparable to using the DOC format of MS WORD for creating documents and that it is not an archivable format. There is a great risk that much of this data will not be accessible anymore after some time and that it will cost much time and money to convert this data to make it usable by others.

In the ECHO (European Cultural Heritage Online) project we had to create an interoperable metadata domain covering ten repositories. Most of them are using relational databases for the reasons mentioned above. These database systems all stated to have an export function to XML, but ALL of the exported “XML”-files still required manual inspection and extra conversions to make them interoperable. The typical problems we found were: (1) Encoding problems: According to the XML header the character encoding was done using UTF-8, but we found out that this was not true and not consistent. (2) Incompatible or wrong use of XML. E.g. the XML-headers were not correct or even the output was not well-formed according to the XML specifications. (3) No XML Schema available: No explicit schema is provided such that validation can not be done. The necessary inspection and conversion costs too much time and money.

[7] XML is hard to read for humans, but at least it is possible. Some are using the term “analyzable” instead of “readable”.

Summarizing, we can say that the use of these very popular relational database systems will probably lead to a loss of data in a few years time.